

# Media

## The preload() Function

The `preload()` function can be used to load any media assets that need to be completely loaded before the `setup()` and `draw()` functions run.

```
let img;

function preload() {
  img = loadImage('myImage.jpg');
}

function setup() {
  // Image is completely loaded when
  the setup function runs
}
```

## Loading Images

The `loadImage()` function is used to load an external image into a p5.js sketch. The function takes one argument for the path to the image as a string.

```
// Loads sky.jpg located in the same
directory as the p5.js sketch file
let skyImage = loadImage('sky.jpg');
// Loads ocean.jpg located inside the
images folder
let oceanImage =
loadImage('images/ocean.jpg');
```

## Drawing Images to the Canvas

The `image()` function allows you to draw images to the canvas. It requires the image element created with the `loadImage()` function and an x and y pixel location to draw the image onto the canvas. Optionally, you can supply the `image()` function with two more arguments that resize the image to a specified width and height on the canvas.

```
function draw(){
  // Draws the image at position (x,y)
  image(img, 0, 0);

  // Draws the image at position (100,
  200) and scales it to 640px and 480px
  image(img, 100, 200, 640, 480);
}
```

## Loading Videos

The `createVideo()` function can be used to load an external video asset into a p5.js sketch. It requires an argument for the path to the video file as a string. Once called, `createVideo()` displays the loaded video onto the webpage as an HTML video element, outside of the canvas.

```
// Loads surfing.mp4 located in the same
// directory as the p5.js sketch file
let surfingVideo =
createVideo('surfing.mp4');
// Loads cooking.mp4 located inside the
// videos folder
let cookingVideo =
createVideo('videos/cooking.mp4');
```

## Looping Videos

The `.loop()` method loops a video. When called, the video will play and repeat on loop infinitely until otherwise interrupted.

In the code example above, the `.loop()` method is called on the `video` variable in the `setup()` function, which starts looping the **myVideo.mp4** file soon after the canvas is created.

```
let video;

function preload(){
  video =
createVideo('videos/myVideo.mp4');
}

function setup(){
  createCanvas(640, 480);
  // Loop myVideo.mp4 file stored in the
  // video variable
  video.loop();
}
```

## Playing Videos

The `.play()` method plays a video. When called, the video will play once and stop when it is finished.

In the code example above, the `.play()` method is called on the `video` variable in the `setup()` function, which plays the **myVideo.mp4** file once soon after the canvas is created.

```
let video;

function preload() {
  video =
  createVideo('videos/myVideo.mp4');
}

function setup() {
  createCanvas(640, 480);
  // Play myVideo.mp4 file stored in the
  video variable
  video.play();
}
```

## Hiding Videos

Use the `.hide()` method to hide the HTML video element that appears on the webpage after loading an external video. When called, it prevents the HTML video element from being visible but does not remove it from the page entirely.

In the code example above, the `.hide()` method is called on the `video` variable in the `setup()` function, which prevents the HTML video element for **myVideo.mp4** file from appearing on the web page.

```
let video;

function preload() {
  video =
  createVideo('videos/myVideo.mp4');
}

function setup() {
  createCanvas(640, 480);

  // Hide the HTML video element
  video.hide();
  // Play myVideo.mp4 file stored in the
  video variable
  video.play();
}
```

## Stopping Videos

To stop a video, use the `.stop()` method. This method will return the video to the starting frame after stopping it—if the video is played again later, it will start from the beginning.

The above code example uses the `videoIsPlaying` variable to keep track of whether `video` is playing or not. When the mouse is pressed, if `video` is currently playing, `video` is stopped. If `video` is not currently playing, `video` is played.

```
let video;

let videoIsPlaying = true;

function mousePressed() {
  // Check if video is playing
  if(videoIsPlaying){
    // Stop video if video is currently
    playing
    video.stop();
    // Set videoIsPlaying to false, as
    video has been stopped
    videoIsPlaying = false;
  }else{
    // Play video if video is not
    currently playing
    video.play();
    // Set videoIsPlaying to true, as
    video has started playing
    videoIsPlaying = true;
  }
}
```

## Filters

The `filter()` function applies a filter to the canvas. It must be provided with the filter type as an argument. Some filter types may require an additional numerical argument.

Filters can also be applied to individual image elements, but not individual video elements, using the `.filter()` method.

```
function draw() {
  // Applies a grayscale filter to the
  canvas
  filter(GRAY);
  // Applies a posterize filter to an
  image
  img.filter(POSTERIZE, 3);
}
```

## The `get()` Function

The `get()` function can be used to access a specific pixel color in the canvas. To do this, the function takes in two arguments for the x and y locations of the pixel and returns the pixel's color as an array of 4 RGBA (red, green, blue, alpha) values.

Alternatively, `get()` can be used to return a rectangular region of the canvas as an image element. When provided with two additional arguments for the width and height, the `get()` function returns the region that starts at the given x and y locations and is bounded by the width and height. When provided no arguments, it returns the entire canvas as an image element.

You can also use the `.get()` method on individual image and video elements to access pixel colors and rectangular regions within the images and videos.

```
// Gets the pixel color at location (200, 100) of the canvas
```

```
let pixelColor = get(200, 100);
```

```
// Gets the rectangular region of the canvas starting at (100, 200) and bound by width of width / 2 and height of height / 2 as an image
```

```
let selectedArea = get(100, 200, width / 2, height / 2);
```

```
// Gets the entire canvas as an image
```

```
let canvasAsImage = get();
```

```
// Gets the pixel color at location (300, 350) of an image
```

```
let imgPixelColor = img.get(300, 350);
```

## The `set()` Function

The `set()` function can be used to modify the color of a pixel at a specific location on the canvas. It requires arguments for the x and y locations of the pixel to modify, and a color—either as an RGBA color array, a p5.js color object, or a single grayscale value.

To reflect new changes to the canvas made with the `set()` function, you must call the `updatePixels()` function.

Using the `.set()` and `.updatePixels()` methods, you can also modify the pixel colors within individual image and video elements before they are drawn to the canvas.

```
// Sets the pixel at location (100, 210) of the canvas to red
```

```
set(100, 210, [255, 0, 0, 255]);
```

```
// Reflects that change in the canvas
```

```
updatePixels();
```

```
// Sets the pixel at location (340, 280) of the image to black
```

```
img.set(340, 280, 0);
```

```
// Reflects that change in the image element
```

```
img.updatePixels();
```

```
// Draws the modified image to the canvas
```

```
image(img, 0, 0);
```

## The `pixels` Array

The `pixels` array is a representation of the pixels that make up a canvas, image, or video frame.

You can access the pixels in the canvas by referring to the `pixels` variable. To access the pixels in an image or video element, use the `.pixels` property.

```
// Logs the pixels array of the canvas to
the console
```

```
console.log(pixels);
```

```
// Logs the pixels array of an image
element to the console
```

```
console.log(img.pixels);
```

```
// Logs the pixels array of a video
element's current frame to the console
```

```
console.log(video.pixels);
```

## Structure of the `pixels` Array

The `pixels` array is formed by starting at the top-left corner of an image, video frame, or canvas. Then, moving down through each row, from left to right, the RGBA values of each pixel are sequentially stored into a single, flat array.

Because each pixel has 4 values (red, green, blue, and alpha), the total length of the `pixels` array can be described by the formula `width * height * 4`, assuming the p5.js sketch has a pixel density of 1. The `width` and `height` variables in the formula describe the pixel dimensions of the image, video, or canvas.

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

```
0, 0, 255, 255,
```

r g b a

```
38, 0, 255, 255,
```

r g b a

```
255, 0, 255, 255
```

r g b a

## Pixel Density

The pixel density in a p5.js sketch is affected by the display density of the computer monitor on which it is viewed. Higher-resolution computer monitors will typically result in greater pixel densities.

To view the current pixel density of a p5.js sketch, use the `pixelDensity()` function with no arguments.

To set the pixel density of a p5.js sketch to a new value, use the `pixelDensity()` function with the new density as its numerical argument.

```
function setup() {
  // Gets the current pixel density
  let density = pixelDensity();

  // Sets the pixel density to 1
  pixelDensity(1);
}
```

## The `loadPixels()` Function

In order to access the `pixels` array of the canvas, you must call the `loadPixels()` function first.

Likewise, before accessing the `.pixels` arrays of image or video elements, you must use the `.loadPixels()` method on the respective element.

```
// Loads pixels array of the canvas
loadPixels();
// Logs canvas' pixels array to the
console
console.log(pixels);
```

```
// Loads pixels array of an image
img.loadPixels();
// Logs image's pixels array to the
console
console.log(img.pixels);
```

## Manipulating the `pixels` Array

You can access and set pixel colors in a canvas by indexing into the individual RGBA values within the canvas' `pixels` array.

To access and set pixel colors in an image or video, index into the individual RGBA values within the `.pixels` array of the image or video element.

```
// Iterates across each pixel in the
canvas
for (let y = 0; y < height; y++) {
  for (let x = 0; x < width; x++) {
    // Gets the index of the red value
    for this pixel
    let indexOfRed = (x + y * width) *
4;
    // Prints this pixel's red value
    console.log(pixels[indexOfRed]);
    // Changes this pixel's color to
    black
    pixels[indexOfRed] = 0; // Red value
    pixels[indexOfRed + 1] = 0; // Green
    value
    pixels[indexOfRed + 2] = 0; // Blue
    value
    pixels[indexOfRed + 3] = 255; //
    Alpha value
  }
}
```

## The `updatePixels()` Function

To reflect any changes to the `pixels` array of a canvas, use the `updatePixels()` function.

To reflect any changes to the `.pixels` array of an image or video element, use the `.updatePixels()` method before drawing the image or video to the canvas.

```
// Loads pixels of the canvas
loadPixels();
// Sets the alpha value of the pixel at
(0,0) to zero
pixels[3] = 0;
// Reflects the change to the canvas
updatePixels();
```

```
// Loads pixels of an image
img.loadPixels();
// Sets the alpha value of the pixel at
(0,0) to zero
img.pixels[3] = 0;
// Reflects the change to the image
element
img.updatePixels();
// Draws the image to the canvas
image(img, 0, 0);
```

 **Print**    **Share** ▼